Stochastic Models and Optimization

Portfolio Selection with Approximate Dynamic Programming

Fischer, Kuo, Marschall, Mirsadeghi

Winter 2018

Contents

1 The Portfolio Selection Problem

1.1 Motivation

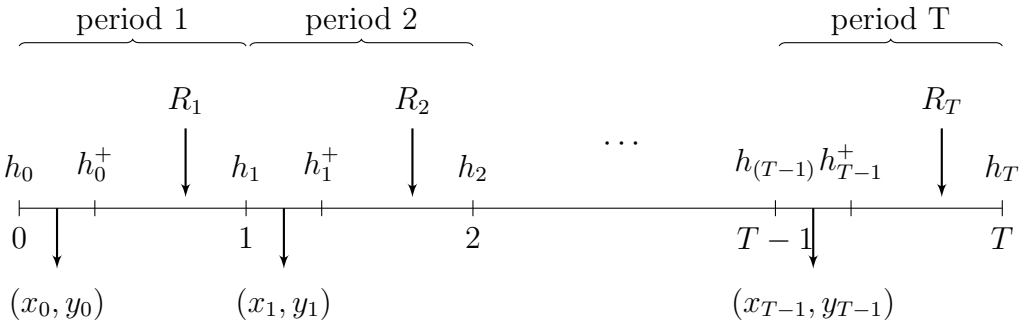
The aim of portfolio selection problem is selecting an investment strategy that maximizes the portfolio return, while minimizing the total risk of the investment. The strategy consists of selecting a subset of a set of tradable entities, like stocks, bonds, commodities and currencies and decide the time to buy or sell those assets. This subject is of main interest for banks and asset management firms.

Traditionally, the portfolio optimization implements Markowitz Portfolio which aims at maximizing the expected return while controlling volatility. In this method no notion of "dead-line" exists so in a sense this method is a single-time period strategy. But in the real world, the investment strategy depends heavily on the time horizon of the investment. For example, banks suggest safer assets for older clients and riskier assets to younger, more stable customers. So the risk aversion depends on the time length of the investment. For example it is reasonable to invest in riskier assets at the beginning of the investment period and toward the end of the investment period move toward safer assets. To take this aspect into account in Markowitz portfolio selection, pre-determined investment limits are set on each tradable assets which cannot be exceeded. But by using the dynamic programming approach, because of considering successive rebalancing periods, it is possible to capture the risk aversion aspect of portfolio optimization problem without pre-determining investment limits of each asset. In this way at the end of each rebalancing time period ("epoch"), depending on whether we are far or close to the investment dead-line, the portfolio adopts itself naturally to adjust the risk aversion and maximize the final expected return.

The issue with using dynamic programming for portfolio optimization the "curse of dimensionality". Even considering discrete states, actions, the dimensions of their spaces is huge and it is impractical to use exact dynamic programming. To overcome this issue, we used Monte-Carlo technique and piecewise linear value functions to turn this problem into a stochastic dynamic programming. This project is an implementation of the stochastic approximated dynamic programming (ADP) in portfolio selection based on the model developed by Karamanis [1] and Berthe [2]. The strategy is tested on weekly data of a selection of stocks available in S&P500.

1.2 Dynamic Programming Formulation

1.2.1 Timeline



The diagram above summarizes the chronology of our problem. We have a total of T periods, and in each period we have 4 elements: h_t - the pre-decision state, (x_t, y_t) - the buyings and sellings, h_t^+ - the post-decision state, and R_t - the realized returns. In the proceeding sections we will explain each element in further detail.

1.2.2 States

In our problem, we consider a total of N different assets and 1 risk-free asset indexed by 0 (which we will call cash). The state of the system is characterized by the holdings we have in each stock n at the corresponding time period t. We focus on the **post-decision state**, $\mathbf{h_t^+} = (h_{0t}^+, h_{1t}^+, ..., h_{Nt}^+) \in \mathbb{R}_+^{N+1}$.

1.2.3 Actions

At each period, we choose a vector of buyings, $\mathbf{x_t} = (x_{1t}, ..., x_{Nt}) \in \mathbb{R}_+^N$ and a vector of sellings, $\mathbf{y_t} = (y_{1t}, ..., y_{Nt}) \in \mathbb{R}_+^N$ to rebalance our portfolio.

1.2.4 Uncertainty

The vector of returns, $\mathbf{R_t} = (R_{0t}, R_{1t}, ..., R_{Nt}) \in \mathbb{R}_+^{N+1}$ at each time period is uncertain. R_{it} is defined as the gross returns on asset i at period t. For example, a return of $+2\%$ is equivalent to a gross return of 1.02.

1.2.5 Constraints

- **Holding constraints**
  In our formulation, we maintain that all asset holdings must be positive. That is, there is no possibility of shorting. This can be equivalently formulated as the following inequality:

$$-x_{it} + y_{it} \leq h_{it} \quad \forall i \in (1, ..., N)$$

  The above inequality says that the amount we sell minus the amount we buy must be less than the amount we have, i.e. we cannot sell more than we have.

- **Budget constraints**
  We include in our model **transaction costs** - costs of buying or selling currencies that are proportional to the amount of assets we are trading. For simplicity, we assume that these transaction costs are the same for buying and selling, and the same for all different assets. We can express the budget constraint as follows:

$$(1+\theta)\sum_{i=1}^{N} x_{it} - (1-\theta)\sum_{i=1}^{N} y_{it} \leq h_{0t}$$

  This inequality expresses that the amount we spend on buying assets (sum of assets bought, plus transaction cost) cannot be more than the current cash we have, plus however much we earn by selling assets (sum of assets sold, minus transaction cost).

- **Positive buying and selling vectors**
  We require $x_{it} \geq 0$ and $y_{it} \geq 0$ for all i and t.

1.2.6 Dynamics

For $1 \leq t \leq T-1$, the following equations characterize the dynamics (separately for the cash asset and the other assets):

$$\begin{cases} h_{0t}^{+} = R_{0t}\left(h_{0(t-1)}^{+} - (1+\theta)\sum_{i=1}^{N} x_{it} + (1-\theta)\sum_{i=1}^{N} y_{it}\right) \\ h_{it}^{+} = R_{it}h_{i(t-1)}^{+} + x_{it} - y_{it}, \quad i \in (1,...,N) \end{cases}$$

For $t = 0$,

$$\begin{cases} h_{00}^{+} = h_{00}^{+} - (1+\theta)\sum_{i=1}^{N} x_{i0} + (1-\theta)\sum_{i=1}^{N} y_{i0} \\ h_{i0}^{+} = h_{i(0)}^{+} + x_{i0} - y_{i0}, \quad i \in (1,...,N) \end{cases}$$

For $t = T$,

$$h_{iT}^{+} = R_{iT}h_{i(T-1)}^{+}, i \in (0,1,...,N)$$

These relationships can be easily verified using the timeline in 1.2.1.

1.2.7 Rewards

We are not concerned with the value or composition of our portfolio for $0 \leq t \leq T-1$, but only with the composition of the final portfolio at $t = T$. Besides the intuitive measure of the sum of all asset holdings, we also want to consider the risk of the final portfolio. Specifically, we use as a risk measure the **Conditional Value-at-Risk**.

First we need to understand **Value-at-Risk** (VaR), which is characterized by a parameter β. VaR_β of a portfolio is equal to the worst return of the $\beta\%$ best cases, or equivalently the best return of the $1-\beta\%$ worst cases. However, the limitation of this risk measure is that it does not take into account the distribution of returns among the $\beta\%$ of worst cases. As such, we use the **CVaR** of our final portfolio h_T^+, which is the expected return given that we stay within the $1-\beta$ worst cases:

$$CVaR_\beta(h_T^+) = -\mathbb{E}[R_{h_T^+} | R_{h_T^+} < -VaR_\beta]$$

We also parametrize the risk-aversion of our investor by the parameter γ. (A more risk-averse investor has a lower γ.) The reward associated with our final portfolio h_T^+ is:

$$\gamma \sum_{i=0}^{N} h_{iT}^+ - (1-\gamma) CVaR(h_T^+)$$

1.2.8 Initialization

We write $V_t^+(h_t^+)$ for the reward-to-go function (from here on value functions) from period t to period T. As such, we initialize the dynamic programming algorithm in the following manner:

$$V_T^+(h_T^+) = \gamma \sum_{i=0}^{N} h_{iT}^+ - (1-\gamma) CVaR(h_T^+)$$

1.2.9 Recursion

From h_{T-1}^+ onwards, we do not have any more decisions to make. Hence

$$V_{T-1}^+(h_{T-1}^+) = \mathbb{E}_{R_T}[V_T^+(h_T^+)|h_{T-1}^+]$$

For $1 \leq t \leq T-2$,

$$V_{t-1}^+(h_{t-1}^+) = \mathbb{E}_{R_t}[\max_{(x_t, y_t)} V_t^+(h_t^+)|h_{t-1}^+]$$

For $t = 0$, we do not have returns available. Hence

$$V_0(h_0) = \max_{(x_0, y_0)} V_0^+(h_0^+)$$

2 Approximate Dynamic Programming

2.1 Motivation

Although the problem has been formulated as an exact dynamic programming problem, it is computationally intractable to solve exactly. This is because of the continuous state space and action spaces (belonging to $\mathbb{R}_+^{N+1}$ and $\mathbb{R}_+^N$ respectively). Even if we decide to discretize and bound the spaces by some limits, it would still be a massive problem to compute the value functions at each period through a backward procedure.

Additionally, we do not know the exact distributions from which the return vectors are drawn. This makes it impossible to maximize our value functions over the expected returns.

2.2 Approximations Made

2.2.1 "Monte Carlo" Simulation

In the exact DP problem, we need to solve the following optimization problem for each time epoch t such that $0 \leq t \leq T-2$:

$$V_t^+ = \max_{(x_t, y_t)} \mathbb{E}_{R_t}[V_{t+1}^+ | h_t^+]$$

However, the distribution of the return matrix R_t is not known to us. As such, we instead generate samples of this return matrix from known asset data. We obtain R_t^s for each training iteration s, and solve instead the following problem:

$$V_t^+ = \max_{(x_t, y_t)} [V_{t+1}^+ | h_t^+, R_t^s]$$

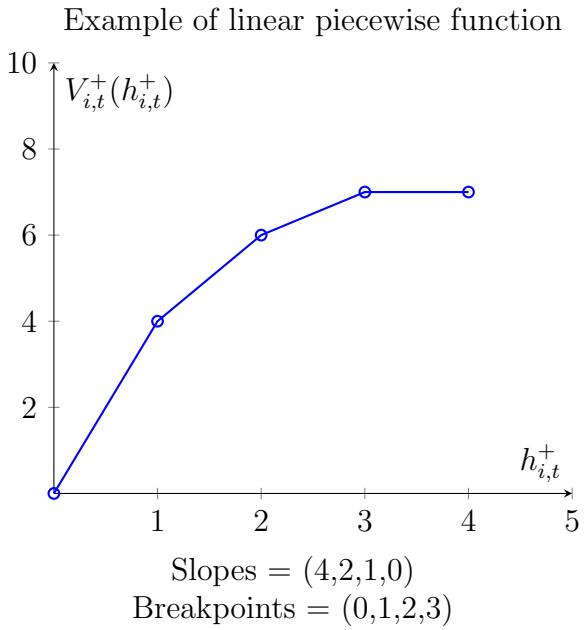
Since we generate many samples of R_t (as many as the number of training iterations we perform), this is similar to a Monte Carlo procedure where we approximate the expectation of a random variable by averaging many samples of it.

2.2.2 Restricting value function space

Instead of searching in the entire function space for exact value functions (reward-to-go functions), we parametrize the function space and focus only on a class of separable linear piece-wise functions. By separability we mean:

$$V_t^+(h_t^+) = \sum_{i \in (0,\ldots,N)} V_{i,t}^+(h_{i,t}^+)$$

By linear piecewise we mean that each function is characterized by a sequence of slopes and breakpoints.



We choose linear piecewise functions to approximate **utility functions**. That is, $V_{i,t}^{+}(h_{i,t}^{+})$ tells us how much we value $h_{i,t}^{+}$, amount of holdings in stock i at time t.

From economic theory, we know that these utility functions should be **increasing** and **concave**. Increasing utility functions just means that we prefer having more of a stock to having less of it, and concave utility functions means that we are risk-averse. These considerations impose the following constraints on our value functions:

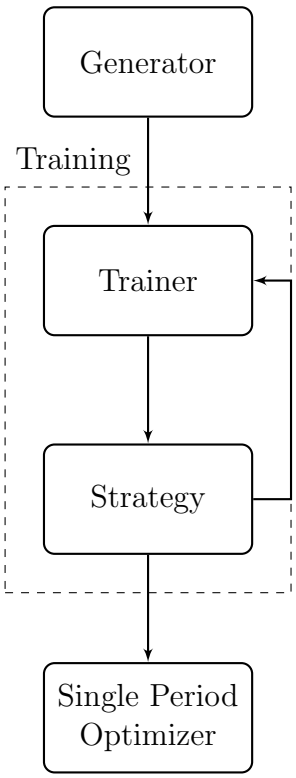
1. Slopes should be ≥ 0.

2. Sequence of slopes should be decreasing in magnitude.

Additionally, we also assume that $V_{i,t}^{+}(0) = 0$.

2.3 Training Procedure for Value Functions

We implement code to train the value functions of our model. The following diagram explains the overall structure of the code:



2.3.1 Generator

For generating the return matrix we take the returns of stocks of nine companies represented in the S&P500 index over the period of 18 years, starting in January 2000. Trying to diversify the possible choices of assets, we specifically selected companies from different industries, whose stock value should not be strongly correlated. The list of the selected companies is

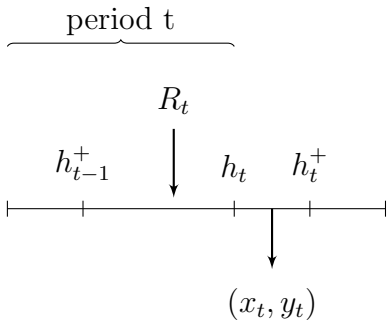
1. Amazon
2. Bank of New York
3. Barclays
4. Coca Cola
5. lowes Companies
6. Nike
7. Norfolk Southern Railway
8. State Street Corporation
9. Walmart

In addition to these stock, one risk-less asset is also included in our model which represent saving in the bank with average return rate of 0.01% per week. We decided to use the weekly instead of daily returns for training our model, since we need some volatility in order to have gradients significantly above zero. In order to maximize the training iterations we will use every

possible set of 52 consecutive observations (simulating a year) in our dataset.

Doing this gives us 844 (slightly) different sets to train our model on, which we then test on the year 2017 trying to check the performance of the portfolio selected by our approximated dynamic programming method.

2.3.2 Single Period Optimizer



This function solves the above single period problem. Given the post-decision state from the previous period h^+_{t-1}, and the vector of realized returns R_t, we want to find the optimal buyings and sellings (x_t, y_t) that will maximize the value of our post-decision state at h^+_t, which is $V^+_t(h^+_t)$. That is, this function solves the following linear optimization problem:

$$\max_{(x_t,y_t)} \left[\sum_{i=1}^{N} V^+_{it}(R_{it}h^+_{i(t-1)} + x_{it} - y_{it}) + V^+_{0t}\left(R_{0t}h^+_{0(t+1)} - (1+\theta)\sum_{i=1}^{N} x_{it} + (1-\theta)\sum_{i=1}^{N} y_{it} \right) \right]$$

subject to the constraints explained in section 1.2.5. We solve this optimization problem using Gurobi, a mathematical programming solver that works with Python.

Additionally, the single period optimizer also returns a **gradient vector**, which will be used by the trainer to update the value functions. The gradient vector tells us how much the value function $V^+_t(h^+_t)$ changes when we increment h^+_t by 1 in a component-wise fashion. To compute this vector, we increase the first component of h^+_t by 1, recompute the above linear optimization problem, obtain the new objective function value and subtract from it the original objective function value to get ΔV_{1t}. We place this ΔV_{1t} in the first component of the gradient vector. This procedure is repeated for all N assets, including the riskless asset (cash), and stacking the results gives us our gradient vector.

2.3.3 Strategy

The strategy function takes in the starting wealth of the investor at period 0, a matrix of returns for each asset from $t = 0$ to $t = T$ and the current value functions. It returns the optimal portfolio as well as gradient matrix for all periods and all assets. This is achieved through a forward process - Strategy calls repeatedly the Single Period Optimizer function, using the optimal portfolio obtained in the previous period as the input for the next period.

2.3.4 Trainer

The trainer function is the main workhorse of this model. It takes as inputs the outputs from the Strategy function and uses it to update all the value functions.

As explained in section 2.1.2, each value function is characterized by 2 arrays, a **breakpoint** array and a **slopes** array. For each training iteration $s = 1, ..., S$, for each V_{it}^{+},

1. Updating breakpoints
   If the optimal h_{it}^{+} is not in the current array of breakpoints, add it to the array. Else, we do not update the breakpoint array.

2. Updating slopes
   We only update one slope for each V_{it}^{+} at each training iteration. The slope we update is the one to the right of h_{it}^{+}. We update the slope $u_{it}^{s,k}$ (slope of i-th asset at time t, k is the index of the slope being updated, and we are at iteration s) using the following formula:

   $$u_{it}^{s,k} = (1 - \alpha^{s})u_{it}^{s-1,k} + \alpha^{s}\Delta V_{it}^{s}$$

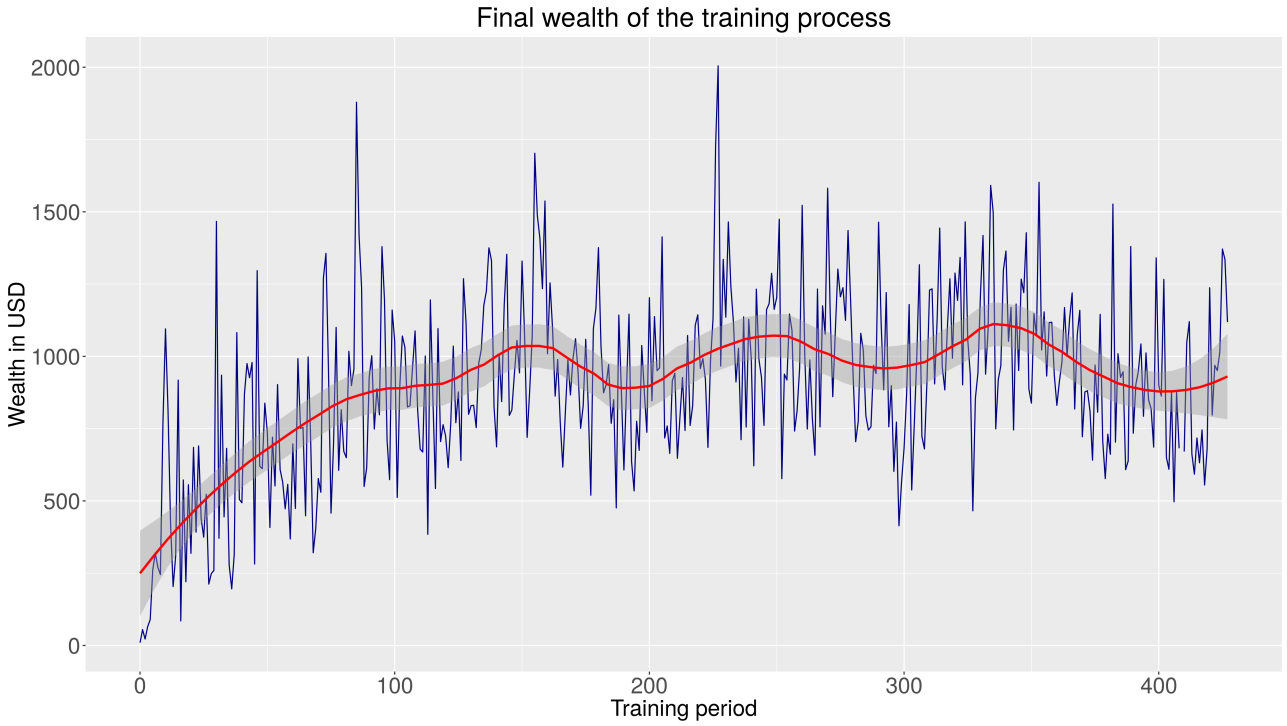
   where ΔV_{it}^{s} is taken from the gradient matrix (output of Strategy).
   We take α, the step-size parameter, to be $\frac{k}{k+s}$ so that the step size decreases as the training iteration increases.

   Additionally, we need to ensure that our value functions are still concave after updating the slopes. This is achieved by averaging slopes and deleting breakpoints as necessary - for details, see the file trainer.py.

3 Results

3.1 Training Phase

Improvement in final wealth over training iterations



The plot depicts the final wealth after every training iteration. The model is trained with 428 sets of 3-year-long return data of the selected stocks. Generally, the final wealth values are very volatile over the training periods. However, an improvement during the first 150 periods (around 3 years) is clearly visisble. The optimization procedure is capable to steadily improve the average final wealth over these first periods. The final wealth remains volatile over the whole training set but on a higher level. After the first 150 periods, a different pattern is observable. The final wealth is on average fluctuating with increasing and decreasing periods. It looks like the model does not gain much from more training periods after a initial learning rate of roughly 150 periods. Despite that the overall trend still seems to be slightly increasing the fluctuations are probably solely driven by the stochastic process of the returns, the general underlying volatility of the market.

Example of trained value function

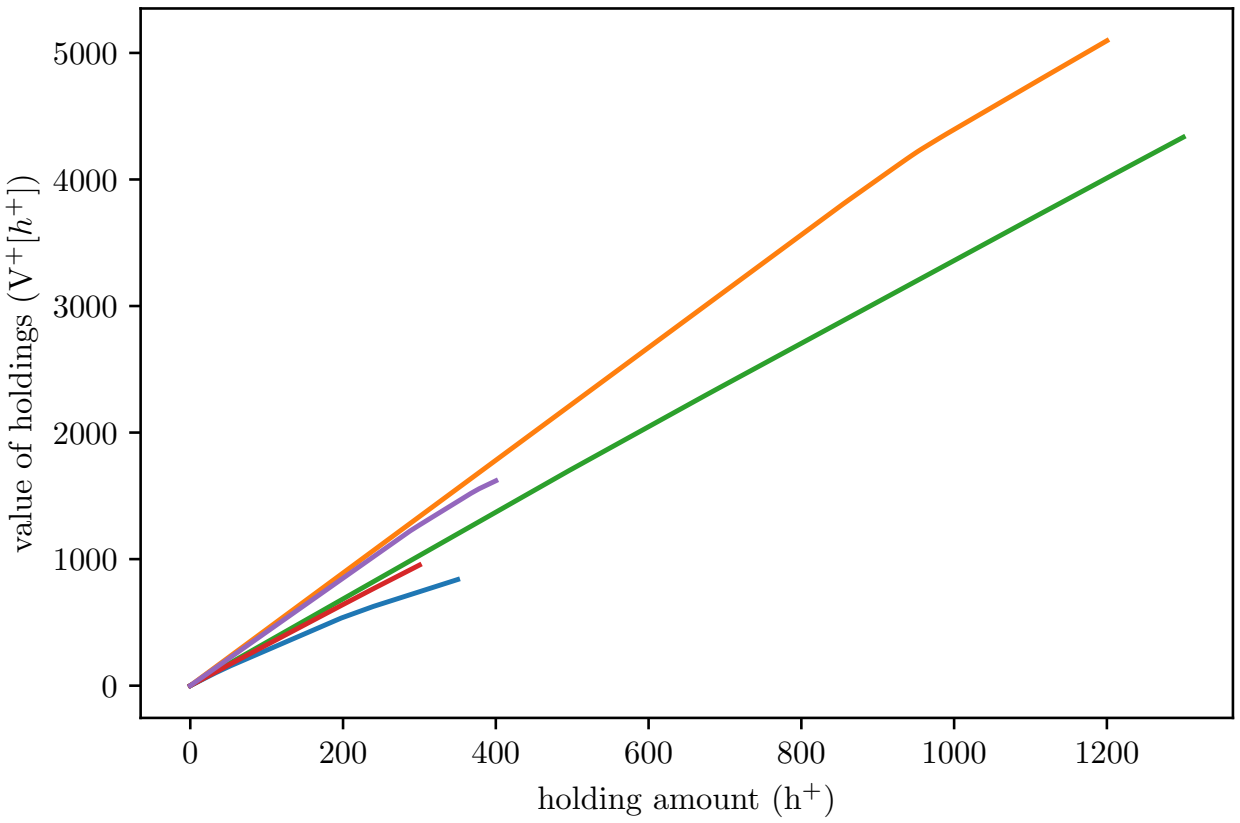


Figure 3.1: Final component value functions (for 5 stocks) after training

The model is initialized with identical slopes for all assets in all periods. After training, we observe differing slopes for different assets - indicating that we get differing amounts of utility for holding the same amount of different assets. This allows the optimizer to choose the 'best' assets to invest in at any given time period. Also, we observe that different breakpoints have been added to the value functions. Slopes on the right are lower, indicating that concavity has been preserved.

3.2 Testing Phase

After training our model on 428 sets of 3-year-long data, it is tested on the next 3 year period. We start with an initial wealth of $1000 in the bank.

Optimal holdings at each period

At the end of 3 years, the total sum of our holdings is $1205. This is significantly better than what we would achieve by leaving the money in the bank ($1016).

Also, we observe that nearer the beginning of the 3 year window, we invest in stocks 1, 2, 3, and 9. Towards the middle, we only invest in stocks 1 and 2. Nearing the end, we only invest in stock 1. The investment behaviour of our model clearly changes over time. The behaviour of the model is mainly driven by the return of the stocks but another factor is also

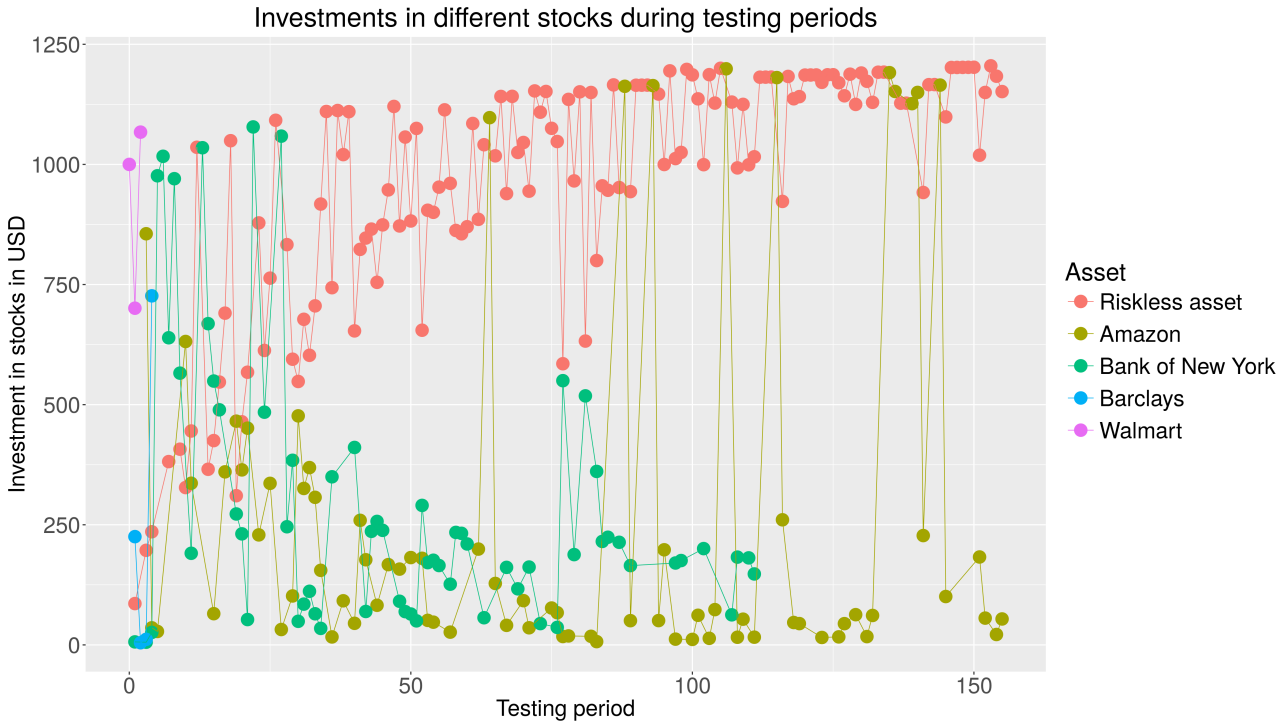


Figure 3.2: The amount of investment in 5 of the stocks considered for model training. Once the holding of any of the stocks reaches zero, its data points are removed in the plot for clarity.

the riskiness of the investments measured by the variance of the stock returns. One motivation of the model is that it does not only take into account the return of the portfolio but also its variance and optimizes the portfolio based on a finite time horizon. The model might be trained in a way that it invests in less riskier stocks towards the end of the time horizon which means mainly leaving a large share in the bank. It is a very plausible explanation for the change of the investment behaviour. Many current investment models are not capable to adjust the investment behaviour over time.

Performance measured against S&P500

Figure 3.3 compares the performance of the S&P500 index and our portfolio selection. Depicted is the actual difference of the value of our portfolio versus the relative value of the S&P500 index. The y-axis actually depicts values in USD. Whenever the plot is in the positive domain, our portfolio outperforms the S&P500 and the other way around.

Both hypothetical investments are set to be equal in the beginning but quickly diverge. Our hypothetical portfolio clearly outperforms the S&P500 for the majority of the three-year period. However, at the end of the three year time horizon our portfolio performs slightly worse than an investment in the S&P500 index. Our portfolio, however, was only trained on 9 different stocks out of the S&P500 plus a riskless asset. The selection of the considered stocks is very arbitrary and our decision was merely based on trying to pick companies from different industries. An realistic portfolio selection model should clearly consider a vast majority of available stocks in order to optimize return and performance. Furthermore, the S&P500 index is no investment with a fixed time-horizon, whereas, our portfolio was trained and optimized on a three year period taking into consideration return and variance of the investment options. Our trained model clearly primarily invests into the riskless asset towards the end of the three year period

and almost completely avoids the risky stocks. Therefore, the performance of the portfolio is limited by the relatively low interest rate of the riskless asset and unsurprisingly looses ground in comparison to the S&P500.

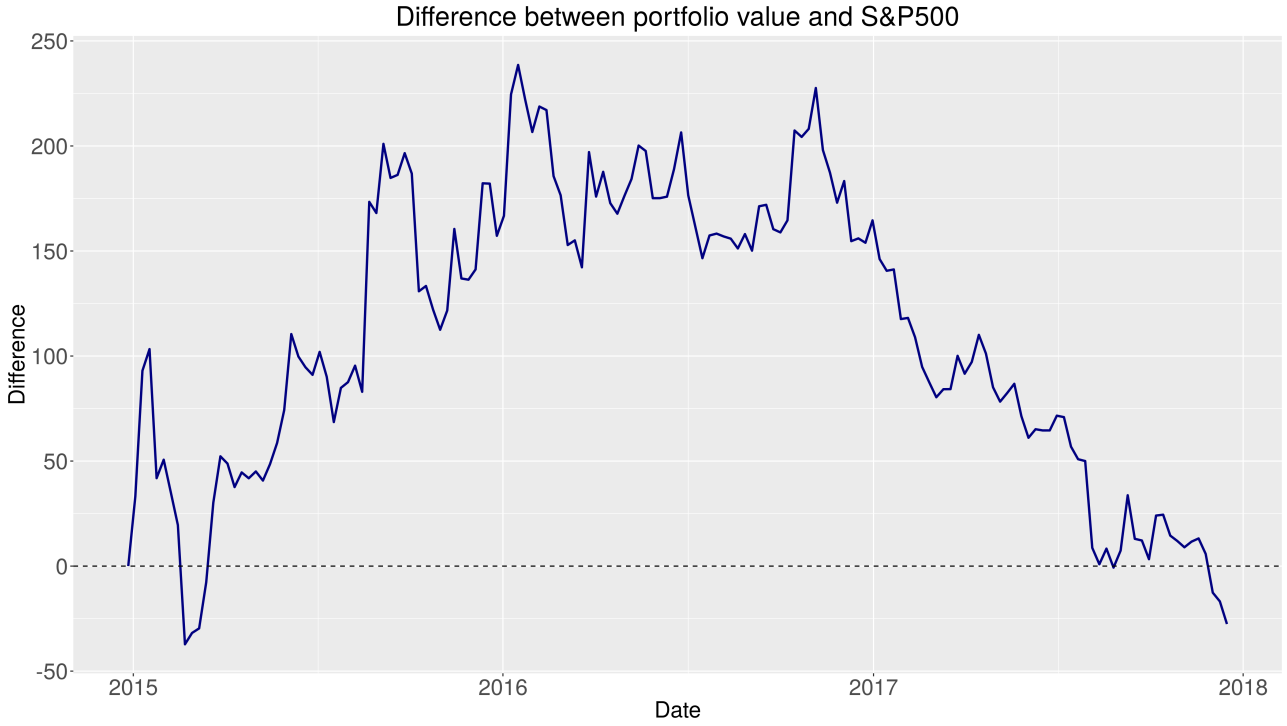


Figure 3.3: The performance of the trained portfolio model versus the S&P500 index on three years of test data

4 Conclusion

This project demonstrates the advantages of dynamic programming as a tool to solve finite horizon problems. Employing the DP algorithm in our portfolio selection problem produces optimal policies that are **time-dependent**, i.e. they are sensitive to the amount of time we have before we have to liquidate all our assets. This is hard to achieve with other portfolio selection tools which produce only stationary policies.

A big limitation of dynamic programming, however, is that it is often computationally intractable and thus difficult to apply to many real-life problems. In this project, we managed to overcome this limitation (to some extent) with suitable approximations.

The results obtained by our model are respectable for the scope of this project. Possible improvements that can be made include training the model on more data, either using more historical data, or developing a model based on historical data that will enable us to generate more training data. However, as pointed out in the previous section the training rate of the model seems to become very flat after roughly 150 training periods which equal 3 years in our application. This finding suggest that adding more training periods would not improve the performance of the portfolio selection model but might instead imposes a general thread of overfitting training data. Alternatively, we could train separate models on bullish and bearish markets, employing them as necessary. In terms of the value functions, it is possible to extend the search to a richer space of functions, possibly allowing for non-linearity. Besides that, the model performance would clearly improve by considering a larger pool of potential assets like stocks, bonds, or commodities.

Due to the limited time and resources we had to settle for selecting only 9 stocks of the American S&P500. Especially interesting would be to consider foreign assets to particularly target and promote risk diversification. The proposed model, however, clearly proves that it is capable to balance between returns and variance of the portfolio and shifts its focus towards less riskier assets towards the end of its pre-defined time horizon. The latter would in practice prove as very valuable advantage especially for private investors with a rather limited time horizon. The proposed model is easily adjustable to account for varying risk perceptions of investors or investment needs. The optimization given a pre-defined time horizon clearly separates the DP-algorithm approach from other portfolio selection models and general economic indices which become more and more popular as alternative capital investments.

Bibliography

[1] Berthe Edouard, *Stochastic Dynamic Programming for Portfolio Selection Problem applied to CAC40*, Jun 2017

[2] Dimitrios Karamanis, *Stochastic Dynamic Programming Methods for the Portfolio Selection problem*, 2013